IIC2343 – Arquitectura de Computadores

Ejercicios Propuestos 1

Profesor: Yadran Eterovic

Ayudantes: Daniel Leal, Jessica Hormazabal, Felipe Valenzuela, Ariel Martinez, Alfonso Irarrazaval, Cristobal Herreros, José Wielandt, Tomás Rivera.

Preguntas

1. Utilice la ISA de la arquitectura MIPS del anexo para escribir un programa que a partir de un número K pueda guardar los subsiguientes n números en los espacios que le siguen a la ubicación de K. Considere que K y n son datos guardados en los espacios de memoria 10 y 9 respectivamente.

2. Modifique una ALU de un bit para que acepte operaciones lógicas, suma y resta con un tercer input. En otras palabras, si se tiene de entradas los inputs A y B, modifique la ALU con tal de aceptar un input C con el que se puedan realizar operaciones tipo $A + B$, $A + C$, entre otros. Puede utilizar todos los componentes y señales que desee.

3. Crea un programa en assembly, utilizando la ISA de la arquitectura MIPS, que multiplique dos números (considere signo).

4. Crea un programa en assembly, utilizando la ISA de la arquitectura MIPS, que eleve un número a otro (aka: a^b).

5. **[TEÓRICO]** En los lenguajes de programación de más alto nivel tenemos la capacidad de pasar parámetros a las funciones que llamamos para que ese ejecuten. Comenta: ¿cómo podríamos hacer lo mismo pero con un programa escrito en assembly?

6. **[TEÓRICO]** ¿Qué propiedades tiene el número 10_b en cualquier base b? ($b > 1, b \in Naturales$)

7. **[TEÓRICO]** ¿Cómo puedo saber cuántos "bits"necesito para representar un natural en cualquier base? ¿Y con un entero?

8. Asuma que $t0$ contiene el valor 0x00101000 ¿Cuál es el valor final de $t2$ después de seguir las siguientes instrucciones?

```
        slt  $t2, $0,  $t0
        bne  $t2, $0,  ELSE
        j    DONE
ELSE:   addi $t2, $t2, 2
DONE:
```

Anexo

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Three register operands
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Three register operands
	add immediate	`addi $s1,$s2,100`	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	`lw $s1,100($s2)`	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	`sw $s1,100($s2)`	Memory[$s2 + 100] = $s1	Word from register to memory
	load half	`lh $s1,100($s2)`	$s1 = Memory[$s2 + 100]	Halfword memory to register
	store half	`sh $s1,100($s2)`	Memory[$s2 + 100] = $s1	Halfword register to memory
	load byte	`lb $s1,100($s2)`	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	`sb $s1,100($s2)`	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immed.	`lui $s1,100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Logical	and	`and $s1,$s2,$s3`	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	`or $s1,$s2,$s3`	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	`nor $s1,$s2,$s3`	$s1 = ~ ($s2 |$s3)	Three reg. operands; bit-by-bit NOR
	and immediate	`andi $s1,$s2,100`	$s1 = $s2 & 100	Bit-by-bit AND reg with constant
	or immediate	`ori $s1,$s2,100`	$s1 = $s2 | 100	Bit-by-bit OR reg with constant
	shift left logical	`sll $s1,$s2,10`	$s1 = $s2 << 10	Shift left by constant
	shift right logical	`srl $s1,$s2,10`	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	`beq $s1,$s2,25`	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1,$s2,25`	if ($s1 != $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1,$s2,$s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for `beq`, `bne`
	`set less than immediate`	`slti $s1,$s2,100`	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than constant
Uncondi-tional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to $ra	For switch, procedure return
	jump and link	`jal 2500`	$ra = PC + 4; go to 10000	For procedure call